(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
9 January 2003 (09.01.2003)

PCT

(10) International Publication Number
WO 03/003688 A2

(51) International Patent Classification[7]: H04L 29/00

(21) International Application Number: PCT/US02/17089

(22) International Filing Date: 29 May 2002 (29.05.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/681,930 27 June 2001 (27.06.2001) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; M/S UPAL01-521, 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventors: PABLA, Kuldipsingh; 3282 St. Ignatius Place #327, Santa Clara, CA 95051 (US). KANUNGO, Rajesh; 765 Limerick Court, Sunnyvale, CA 94087 (US). NARAYANAN, Venkatesh; 40771 Canyon Heights Drive, Fremont, CA 94539 (US).

(74) Agent: PENILLA, Albert, S.; Martine & Penilla, LLP, 710 Lakeway Drive, Suite 170, Sunnyvale, CA 94085 (US).

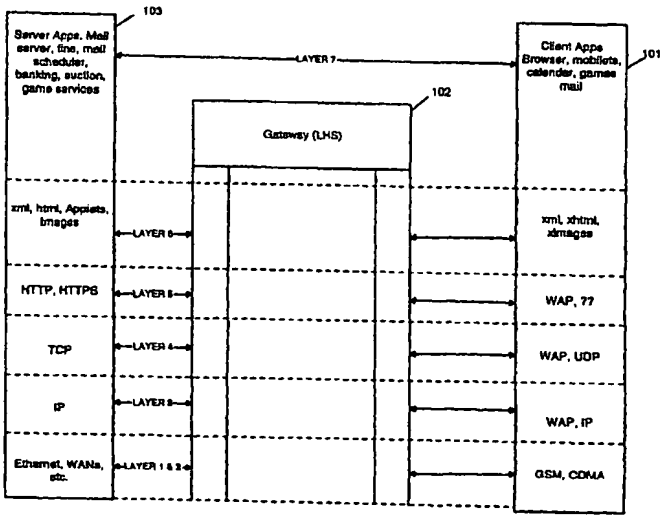
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: APPLICATION FRAMEWORK FOR MOBILE DEVICES



(57) Abstract: An application framework for mobile devices is described comprising a three-tier software architecture for wireless devices to allow high-powered backend services to be accessible by low-powered wireless client devices. The present invention defines a layered end-to-end architecture and an application framework, called mobilet framework, for client devices to allow applications to run on wireless devices in a vendor-neutral and platform independent manner. The wireless device may be viewed as a cache or a viewport through which high-end services can be accessed. The cache may be synchronized periodically with the servers and/or service providers through a gateway portal targeted specifically at low-end wireless devices. The mobilet framework for low-end client devices defines an Application Programming Interface as well as an abstraction for platform independent applications called mobilets.

WO 03/003688 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

APPLICATION FRAMEWORK FOR MOBILE DEVICES

BACKGROUND OF INVENTION

1. Field of the Invention

This invention relates to the field of software architecture for wireless devices. More specifically the invention relates to an application framework for wireless client devices to allow applications to run on these devices in a vendor-neutral and platform independent manner.

2. Background Art

The wireless communication environment is characterized by the existence of multiple commercial networks, such as Mobitex, Cellular Digital Packet Data (CDPD), Global System for Mobile communication (GSM), Radio Frequency (RF), satellite, cellular and/or Wireless

Application Protocol (WAP), XHTML (Extended Hyper Text Markup Language), or Wireless LAN (Local Area Network) networks, and numerous other protocols. Incompatibility between these networks makes it impossible to create common applications for devices that use these protocols. Current systems operate in an end-to-end fashion. That is, services are linked from provider to clients of that provider and are usually independent of other providers.

Another problem is that wireless devices like cellular phones, pagers, Personal Data Assistants (PDA) have very small footprints (i.e. they are small). Thus, they have limited memory, processing capacities, and display size, hence, are limited in the size of applications that they can process. Current systems cannot support multiple applications. For example, some cellular phones have four lines of display and some have up to six. The differing capabilities limits the size of applications that are available for these devices. The proposed application framework makes these limitations transparent. The framework allows service providers to field applications or provide applications to these wireless devices without much knowledge about what these devices are actually capable of.

The following definitions are examples of the various forms of wireless communication protocols. They are not intended to be a complete list of the various protocols used in the wireless communication industry.

CDPD (Cellular Digital Packet Data) is a specification for supporting wireless access to the Internet and other public packet-switched networks. CDPD is an open specification that adheres to the layered structure of the Open Systems Interconnection model and has the ability to be extended in the future. CDPD's support for packet-switching means that a persistent link is not needed. The same broadcast channel may be shared among a number of users at the same time.

GSM (Global System for Mobile) is a digital mobile telephone system that is widely used in Europe and other parts of the world. GSM uses a variation of time division multiple access (TDMA) and is the most widely used of the three digital wireless telephone technologies (TDMA, GSM, and CDMA (code-division multiple access)).

Mobitex is a packet switched system for mobile data communication. This means that all data are transferred over radio waves in customized units or packets. This way, the network is used efficiently, and connection times are very short. One advantage of this is that subscribers only pay for packets of data that are sent and not for the connection time with, for example, a mobile telephone. This means that the system connects senders and receivers wherever they are in the area covered (known as roaming). A great advantage of the Mobitex network is that messages that are sent are coded in a special way so that the network automatically corrects mistakes and requests a re-send. So the receiver can be sure that no distorted or incorrect messages are delivered.

Summary of Invention

An application framework for mobile devices is described. In one embodiment, a three-tier software architecture for wireless devices to allow high-powered backend services to be accessible by low-powered wireless client devices. The present invention defines a layered end-to-end architecture and an application framework for client devices to allow applications to run on these wireless devices in a vendor-neutral and platform independent manner thereby making footprint and protocol restrictions transparent to the client.

In one or more embodiments, a wireless device may be viewed as a cache or a viewport through which high-end services can be accessed. The cache may be synchronized periodically with the servers and/or service providers through a gateway portal targeted specifically at low-end wireless devices. Some of these services may be local, some remote and some split in-between the low-end client and the higher end server. The present mobilet framework for low-end client devices defines an Application Programming Interface (API) as well as an abstraction for platform independent (e.g. Java) applications called mobilets. This framework allows server application and client application interaction on a class of devices in a vendor neutral manner.

Brief Description of Drawings

Figure 1 is a diagram of the en-to-end protocol view for the wireless client, in accordance with one embodiment of the present invention.

Figure 2 is an illustration of the layered structure of the client tier, in accordance with one embodiment of the present invention.

Figure 3 is a state diagram depicting the life of a mobilet in the framework, in accordance with one embodiment of the present invention.

Figure 4 is a block diagram of a processing environment comprising an object-oriented runtime environment capable of providing a suitable software execution environment for an embodiment of the present invention.

Figure 5 is a block diagram of one embodiment of a computer system capable of providing a suitable hardware execution environment for an embodiment of the present invention

DETAILED DESCRIPTION

The invention defines a three-tier software architecture for wireless devices to allow high-powered backend services to be accessible by low-powered wireless client devices. In the following description, numerous specific details are set forth to provide a more thorough description of embodiments of the invention. It will be apparent, however, to one skilled in the art, that the invention may be practiced without these specific details. In other instances, well known features have not been described in detail so as not to obscure the invention.

In general, low-powered wireless devices like cell phones, pagers, and Personal Data Assistants (PDA), have small footprints and communicate using various incompatible protocols. Usually, wireless devices use protocols that are service provider dependent therefore making it difficult to run common applications across services (i.e. protocol). In addition, these devices are limited in display size, memory and processing power. For example, some devices have four lines of display and some have up to six. Moreover, the specifications on wireless devices constantly vary as manufacturers vie to reduce footprint while providing more functionality. This makes it difficult to standardize and provide applications across protocols.

Different service providers use different protocols to communicate with clients on their wireless networks, making it virtually impossible to develop applications that are device independent. This invention defines a framework whereby wireless applications can be run independent of protocol, footprint, and display size. That is, applications developed for this framework will be able to run on any wireless device without prior knowledge of the capabilities of the devices. For purposes of this specification, applications that run on this framework are called mobilets because of their applicability to mobile (i.e. wireless) services.

Traditionally, Internet devices like screen phones and set top boxes have been fat clients. That is, they have a high-end rendering engine and a set of services resident in the devices. This functionality requires the devices to have more memory and more processing power than is cost efficient for small devices.

The present invention describes a three-tier software architecture for wireless devices to allow high-powered backend services to be accessible by low powered wireless client devices. For example, services that are generally available on desktop and similar environments can be made available to the mobile user independent of service provider. The present invention defines a layered end-to-end architecture and an application (i.e. mobilet) framework for client devices to allow applications to run on these wireless devices in a vendor-neutral and platform independent manner.

A wireless device can be viewed as a cache or a viewport through which high-end services can be accessed. The cache may be synchronized periodically with the servers and/or service providers through a gateway portal targeted specifically at low-end wireless devices. Some of these services may be local, some remote and some split in-between the low-end client and the higher end server. The present mobilet framework for low-end client devices defines an Application Programming Interface (API) as well as an abstraction for platform independent (e.g. Java) applications called mobilets. This framework allows server application and client application interaction on a class of devices in a vendor neutral manner.

Layered End-to-End Protocol Architecture

This is a peer-to-peer set of layers defined to optimize definition of services by abstracting out the effects of rapid changes in technology. In one or more embodiments, each

layer of the architecture provides a certain set of services to the upper layer and uses certain services from the layer below.

In one embodiment, the expense of online connectivity for the wireless user forces the focus on offline content accessing with the exception of time-sensitive data (e.g. stock quotes). Thus, the mobile device acts as a cache or reservoir of information that may periodically synchronize with a server to update its cache. Optionally, a push service may send important events to the device. This means that continuous connectivity is not necessary unless time sensitive and real-time information is needed.

Figure 1 shows a diagram of the end-to-end protocol view for the wireless clients. There are seven protocol layers and three service tiers in the model. The model is based on the OSI 7 layer architecture specified in "Computer Networks" by Tanenbaum. The three-tier architecture comprises the client tier, the gateway tier, and the server tier. The client tier, block 101, comprises a KVM (K Virtual Machine) or equivalent virtual machine capable of scheduling device independent applications. The KVM is the small device equivalent of the Java Virtual Machine (JVM). Like JVM, the KVM coexists with the native operating system and other software on the client device. Other Java packages are used to provide an API for Web like (e.g. WAP, XHTML) functionality, sandbox security, a framework for running Java applications, and other services.

The Wireless Gateway tier 102 is responsible for providing services that lighten the load on the client by doing as much preprocessing as possible and for any protocol translation between the server and the client device. For example, the gateway performs content transformation to WML (Wireless Markup Language) or XHTML, converts from HTTP (Hyper Text Transport Protocol) to WAP, does Byte-code verification, authenticates Java applications, provides push services, and other services.

The Server tier 103 comprises a large group of services that may be available on enterprise servers. Some of the services are provider dependent and run on client devices. Examples of services are banking applications, brokerage services, etc. Servers may also use push services to push client applications into the client device making the client applications portals into the services provided by the server.

Each layer of the architecture provides a certain service and the subdivision is arranged to provide certain advantages. For example, different vendors may choose to implement or support different standards for communication with their clients. Client devices may have different capabilities or may use different implementation to provide same functionality. It also allows software to be easily portable between client devices.

Layers 1 and 2 are the physical and data link layers. The connections on the server side (i.e. server to gateway communication) may be through an Ethernet, Wide Area Network (WAN), the Internet, or other similar communication network. The gateway to client side communication may be through any of the available wireless communication protocols such as GSM, CDMA, and TDMA.

Layer 3 is a network layer with IP (Internet Protocol) communication between the server and the gateway. The gateway to client side may use IP or WAP protocol for communication. Layer 4 is the transport layer probably using TCP (Transmission Control Protocol) on the server to gateway side and WAP, UDP (User Datagram Protocol), or TCP on the gateway to client side. The WAP may be more efficient because it allows data for compression, however, most current Web transport services use TCP.

Layer 5 is the session layer involving HTTP, HTTPS (i.e., secure HTTP), and other forms of communication between services on the server to gateway side. WAP may be the most efficient system on the gateway to client side because it has an efficient mechanism for Gets and Sets functions. Layer 6 is the presentation for markup and may use HTML, or XML (Extensible Markup Language) for server to gateway communication. The gateway to client side may use WML (Wireless Markup Language), or XHTML for communication. WML is more than a markup language because it has telephony extensions.

The final layer, 7, is the applications layer. This layer includes preparation of graphical data for presentation, action oriented metaphors, directory services, mail services, and etc. Graphical data between the server and the gateway is presented in a format such as GIF (Graphical Interchange Format) or JPEG. This data is converted in the gateway tier to a format such as WAP compressed 4-bit graphics (i.e. bitmaps) for communication to the client device.

Action oriented metaphors, such as JavaScripts and applets, from the server side are converted by the gateway to WMLScript and mobilets, respectively, before transmission to the client device. For directory services, the gateway acts as proxy to the client tier. Mail services may be sent via standard text paging systems to the client from the gateway.

In this three-tier architecture, the gateway is below the application layer and acts as a general purpose protocol transformation engine. Therefore, the gateway has very little to do with how server applications and client applications interact in a peer-to-peer fashion. The gateway can be used to do bytecode verification and to target client devices belonging to a particular category. The wireless gateway handles communications between server and client in order to accommodate bandwidth restrictions, space restrictions, and security concerns that are specific to wireless devices, and also Internet constraints by providing some kind of barrier and transformation between client and server. For example, the gateway may take Web pages from a server and strip out of the contents some unnecessary information and make it available to wireless phones without any problem.

Authentication, security, and encryption issues on the server to gateway side may be handled using digital certificates, Secure Sockets, Digital Hashes (e.g. MD5), RSA and DES encryption of various strengths.

Client Tier Internal Architecture

The protocol mapping and end-to-end architecture discussed above highlights the difficulty in developing applications based on any particular set of protocols even for the same class of devices. It is harder still for general purpose wireless service providers to support client applications on the vast array of wireless devices even with the help of transformation gateway support since applications do not have access to the same set of local services. The present invention defines local services available on the client device that would allow applications to run provider-neutral and in platform independent manner. A layered architecture is defined that encapsulates protocol and system specific implementation features in abstractions. For example, the client does not have to worry about the markup language (WML or XHTML) or whether or not the protocol engine is implemented in native or Java.

Figure 2 is an illustration of the layered structure of the client tier. The RTOS (Real Time Operating System) layer 202 comprises the wireless small device operating system 224 with its linking and networking APIs block 220. The hot updates object 222 allows updates and installation of new pieces of software on the client device RTOS layer without affecting other layers in the architecture and without the client device requesting for the update. RTOS 224 is generally native code (i.e. device dependent), but may be written in object-oriented language like Java. In one or more embodiments, layers 202, 204, and block 210 may be written in native code.

On top of the RTOS layer 202 is the virtual machine (VM) layer 204. VM layer 204 comprises the K Virtual Machine 206 and system classes 226 through 234. System classes 206 through 226 are integral part of the K Virtual Machine. As discussed earlier, the K Virtual Machine is a small device version of the Java Virtual Machine (JVM). The KVM allows multi-threading in order to make inter-mobilet interaction easy and predictable. Although KVM and JVM are used in this specification, it would be obvious to those of ordinary skills that any virtual machine that performs similar functions can be used instead to provide similar functionality.

The final layer is the application layer 208. This layer contains the platform specific mobilet framework object class 210, the platform independent mobilet framework object class 212, and application object classes 214 through 218. This arrangement allows application objects 214 through 218 to be platform and vendor neutral.

Application objects 214 through 218 are the mobilets. In one or more embodiments, the present invention is used to track shipping packages. For example, assuming FedEx has a shipment for a client, mobilet 216 could be subscribed to during shipping, which would automatically provision (i.e. push out to) the client"s wireless device. Another example is if a client is about to receive a package from FedEx, the recipient"s wireless phone will automatically be provisioned with FedEx mobilet 216 if the sender had provided a phone number during shipping. When the package arrives at the recipient"s door, they will either get a phone call, or mobilet 216 runs and alerts the client of the arrival.

Basically, service providers may have mobilets ready to run on service recipient's wireless devices. The present invention allows service providers like FedEx to alert clients of

important events if the clients have wireless devices that can be provisioned with mobilets. Also, a client sending a package to somebody else may track the package with their cell phone if the cell phone has a tracking mobilet (e.g. FedEx mobilet 216). Similarly, a client using their cell phone can connect to a stock ticker provider to get the current value of stocks. The service provider or ticker provider can push the mobilet required to view the ticker to the wireless device. Thus, the present invention allows wireless device users to subscribe to services on the fly.

The platform handles all communications between the wireless device and the service provider using mobilets that implement user interface functions. Functionally, mobilets would be capable of determining how the platform works, what kind of user interfaces are supported, and the best way to display information. For example, if the device does not have a browser then mobilets handle the browser function.

Available services include subscription, publishing, sinking, etc. A service provider may publish available services and clients can subscribe to available services on the fly. Sinking allows clients that have desktop machines from which they can access e-mail, calendar, and other functions to sink-up their cell phones or wireless devices with their desktop to allow access to those functions from the wireless device.

A service provider may broadcast availability of certain services. Client"s that are interested may pick and choose those services they would like to subscribe to, for example, stock ticker for tracking investments and FedEx mobilet for tracking packages. If a client is not subscribing to FedEx but the client would like to track an incoming package or outgoing package then the client"s service provider can push that mobilet into the client"s wireless device. Another example is that a client may be interested in subscribing to some services available on the Internet.

Mobilet Framework

A mobilet, like an applet, is an application written to the mobilet framework specifications. It resides on top of a thin runtime container (Mobilet framework). Mobilets have a default behavior unless the mobilet developer overrides the APIs. Although mobilets can communicate with each other through the framework, the state of each mobilet is

managed by a mobilet manager. Thus, the mobilet manager manages all the mobilets in the framework. The mobilet manager is responsible for starting, stopping, initializing, suspending, etc. for all mobilets. For example, a mobilet cannot requisition the display screen of the wireless device without permission from the mobilet manager.

Most wireless devices are usually very limited in visual display capability therefore only one application may operate in the foreground. This invention provides a desktop type metaphor that is a desktop kind of feel for applications on the wireless device. This means that the user should be able to switch between applications just like on the desktop. But in general, only one application will be active in the foreground at a time. The remaining applications may be in the background. Other applications may be active in the background so long as they are not consuming much resource. For example, one thread could be waiting on a circuit and when it becomes active, it might try to take the foreground by requesting for access from the mobilet manager.

Each mobilet has an identification (ID) that uniquely refers to it. The mobilet ID may contain references to its name, and other information (e.g. platform dependent messages). The contents of the mobilet ID are generally not visible to the mobilet except for certain method calls. The mobilet manager handles each mobilet without a pointer that way one mobilet cannot interfere in the operations of another mobilet.

The mobilet manager creates a registry of all mobilets in the framework. When a mobilet is started and is initialized, its ID is stored in the mobilet registry. The mobilet manager may then pass an object (e.g. a cookie) to the mobilet so that the mobilet may discover the environment around it. Most of the environment information is stored in the mobilet manager, but a cookie is a safe interaction because it is in standard API, i.e., standard object calls.

The mobilet manager is responsible for giving mobilets life by giving them a mobilet ID and stuffing them in the mobilet registry. The manager is responsible for initializing, stopping, stocking, putting the mobilets in the background. No mobilet function happens directly without permission from the mobilet manager. So if one mobilet wants access to the screen, it must request it through the mobilet manager. If it"s okay (e.g. a higher priority task or the current active task is preemptable), then the manager will shut down the active mobilet

by placing it in the background before bringing the requesting mobilet to the foreground. Examples of higher priority tasks include event messenger and instant messenger services. These services may notify the user and request confirmation whether the user wants to view the messages instantly. However, no mobilet may directly request other mobilet to relinquish access. Access must always be obtained through the mobilet manager, so there is an access control to minimize the possibilities for destructive interaction. For example, in order to notify the user and request confirmation whether or not to view a message, the service must first request access for the screen from the mobilet manager.

The mobilet manager does validation of the mobilet ID with collaboration from the mobilet registry. References to a mobilet are via its ID. The registry is a table of what kind of services are available, i.e., what type of mobilets are available, there capabilities, and what kind of information they contain. For example, the e-mail may want to use a calendar function so it would inquire from the mobilet registry for available services. If there is a calendar function, it may then request, from the mobilet manager, that the calendar function be put in the foreground.

The mobilet manager handles launching of applications (i.e. mobilets), inter-mobilet communication, lifecycle of mobilets, registration of mobilets, the state of each mobilet, user interface (i.e. interaction), etc. Figure 3 is a state diagram of the life of a mobilet. At state 300, the mobilet is initialized; the mobilet manager passes a context (e.g. a cookie) to allow the mobilet to determine its environment. The mobilet manager then creates the mobilet by giving it an ID and publishing it in the registry. After registration is complete, the mobilet may request move to the foreground, if granted, the mobilet is put in state 304, otherwise it is in state 302. At state 304, the mobilet has access to resources like the display, and other user interface components.

If access is not granted to proceed to foreground 304, the mobilet is put in the background state 302. A mobilet can only be destroyed from either the background state 302 or from the paused state 306. The mobilet manager may move the mobilet between the background state 302, foreground state 304, and the paused state 306, depending on priorities and usage requirements. In this fashion, the mobilet manager manages the state of the mobilet once it has been initialized and is in the framework.

Because the framework makes the wireless device act like a cache of services, it allows for download of proxy stubs that convert the wireless device into a service provider. Thus, in the service provider configuration, the wireless device may be used to provide services to other wireless devices, for example. The framework also provides persistent storage for client applications and sandbox security to prevent collision and inadvertent destruction of services.

In one or more embodiments of the present invention, sample Java™language source code implementing the framework and its embedded services are provided in Appendix A.

Embodiment of a Processing Environment

An embodiment of the invention is directed, though not limited, to distributed applications, such as those in which a server application serves one or more wireless client applications. Such systems may be implemented using object-oriented programming environments that produce executable software objects. To facilitate object compatibility between the client and server, the software objects may be implemented in a platform independent manner, or the client and server systems may share common or compatible operating platforms. The clients and server may execute within separate machine or virtual machine runtime environments, within a single runtime environment, or a combination of the foregoing arrangements. The following description refers to an embodiment of a virtual machine-based runtime environment, though it will be obvious that the invention is not limited to such.

Applications typically comprise one or more object classes. Classes written in high-level programming languages, such as the Java™ programming language, may be compiled into machine independent bytecode class files. Alternatively, classes may be compiled into machine dependent, executable program code for direct execution by a given hardware platform. In the machine independent case, each class file contains code and data in a platform-independent format called the class file format. The computer system acting as the execution vehicle contains a program called a virtual machine, which is responsible for executing the code in each class file. (A hardware system may also be used that directly executes bytecode of class files.)In a virtual machine environment, the classes of an application are loaded on demand from the network (stored on a server), or from a local file

system, when first referenced during the application's execution. The virtual machine locates and loads each class file, parses the class file format, allocates memory for the class's various components, and links the class with other already loaded classes. This process makes the code in the class readily executable by the virtual machine.

Figure 4 illustrates the compile and runtime environments for an example processing system. In the compile environment, a software developer creates source files 400, which contain the programmer readable class definitions written in the source programming language, including data structures, method implementations and references to other classes. Source files 400 are provided to pre-compiler 401, which compiles source files 400 into ".class" files 402 that contain bytecodes executable by a virtual machine. Bytecode class files 402 are stored (e.g., in temporary or permanent storage) on a server, and are available for download over a network. Alternatively, bytecode class files 402 may be stored locally in a directory on the client platform.

The runtime environment contains a virtual machine (VM) 405 which is able to execute bytecode class files and execute native operating system ("O/S") calls to operating system 409 when necessary during execution. Virtual machine 405 provides a level of abstraction between the machine independence of the bytecode classes and the machine-dependent instruction set of the underlying computer hardware 410, as well as the platform-dependent calls of operating system 409.

Class loader and bytecode verifier ("class loader") 403 is responsible for loading bytecode class files 402 and supporting class libraries 404 into virtual machine 405 as needed. Class loader 403 also verifies the bytecodes of each class file to maintain proper execution and enforcement of security rules. Within the context of runtime system 408, either an interpreter 406 executes the bytecodes directly, or a "just-in-time" (JIT) compiler 407 transforms the bytecodes into machine code, so that they can be executed by the processor (or processors) in hardware 410.

The runtime system 408 of virtual machine 405 supports a general stack architecture. The manner in which this general stack architecture is supported by the underlying hardware 410 is determined by the particular virtual machine implementation, and reflected in the way

the bytecodes are interpreted or JIT-compiled. Other elements of the runtime system include thread management (e.g., scheduling) and garbage collection mechanisms.

Embodiment of Computer Execution Environment (Hardware)

An embodiment of the invention can be implemented as computer software in the form of computer readable code executed on any computer processing platform, or in the form of software (e.g., bytecode class files) that is executable within a runtime environment running on such a processing platform. An embodiment of the invention may be implemented in any type of computer system or programming or processing environment, including embedded devices (e.g., web phones, set-top boxes, etc.) and "thin" client processing environments (e.g., network computers (NC's), etc.). An example of a general computer system is illustrated in Figure 5. The computer system described below is for purposes of example only.

In Figure 5, keyboard 510 and mouse 511 are coupled to a system bus 518. The keyboard and mouse are for introducing user input to the computer system and communicating that user input to processor 513. Other suitable input devices may be used in addition to, or in place of, the mouse 511 and keyboard 510. I/O (input/output) unit 519 coupled to system bus 518 represents such I/O elements as a printer, A/V (audio/video) I/O, etc.

Computer 500 includes a video memory 514, main memory 515 and mass storage 512, all coupled to system bus 518 along with keyboard 510, mouse 511 and processor 513. The mass storage 512 may include both fixed and removable media, such as magnetic, optical or magnetic optical storage systems or any other available mass storage technology. Bus 518 may contain, for example, address lines for addressing video memory 514 or main memory 515. The system bus 518 also includes, for example, a data bus for transferring data between and among the components, such as processor 513, main memory 515, video memory 514 and mass storage 512. Alternatively, multiplexed data/address lines may be used instead of separate data and address lines.

In one embodiment of the invention, the processor 513 is a SPARC™ microprocessor from Sun Microsystems, Inc. or a microprocessor manufactured by Intel, such as the 80X86,

or Pentium processor, or a microprocessor manufactured by Motorola, such as the 680X0 processor. However, any other suitable microprocessor or microcomputer may be utilized. Main memory 515 is comprised of dynamic random access memory (DRAM). Video memory 514 is a dualvideo random access memory. One port of the video memory 514 is coupled to video amplifier 516. The video amplifier 516 is used to drive the cathode ray tube (CRT) raster monitor 517. Video amplifier 516 is well known in the art and may be implemented by any suitable apparatus. This circuitry converts pixel data stored in video memory 514 to a raster signal suitable for use by monitor 517. Monitor 517 is a type of monitor suitable for displaying graphic images. Alternatively, the video memory could be used to drive a flat panel or liquid crystal display (LCD), or any other suitable data presentation device.

Computer 500 may also include a communication interface 520 coupled to bus 518. Communication interface 520 provides a two-way data communication coupling via a network link 521 to a local network 522. For example, if communication interface 520 is an integrated services digital network (ISDN) card or a modem, communication interface 520 provides a data communication connection to the corresponding type of telephone line, which comprises part of network link 521. If communication interface 520 is a local area network (LAN) card, communication interface 520 provides a data communication connection via network link 521 to a compatible LAN. Communication interface 520 could also be a cable modem or wireless interface. In any such implementation, communication interface 520 sends and receives electrical, electromagnetic or optical signals which carry digital data streams representing various types of information.

Network link 521 typically provides data communication through one or more networks to other data devices. For example, network link 521 may provide a connection through local network 522 to local server computer 523 or to data equipment operated by an Internet Service Provider (ISP) 524. ISP 524 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 525. Local network 522 and Internet 525 both use electrical, electromagnetic or optical signals which carry digital data streams. The signals through the various networks and the signals on network link 521 and through communication interface 520, which carry the digital data to and from computer 500, are exemplary forms of carrier waves transporting the information.

Computer 500 can send messages and receive data, including program code, through the network(s), network link 521, and communication interface 520. In the Internet example, remote server computer 526 might transmit a requested code for an application program through Internet 525, ISP 524, local network 522 and communication interface 520.

The received code may be executed by processor 513 as it is received, and/or stored in mass storage 512, or other non-volatile storage for later execution. In this manner, computer 500 may obtain application code in the form of a carrier wave. Application code may be embodied in any form of computer program product. A computer program product comprises a medium configured to store or transport computer readable code or data, or in which computer readable code or data may be embedded. Some examples of computer program products are CD-ROM disks, ROM cards, floppy disks, magnetic tapes, computer hard drives, servers on a network, and carrier waves.

Program Listing Deposit

```
public abstract interface MobiletID {
String getName();
boolean isActive();
boolean isEqual(MobiletID other);
 Mobilet getMobilet();
void setContext(MobiletContext ctx);
MobiletContext getContext();
}
public abstract interface Mobilet extends Xlet {
// inherits destroyXlet, initXlet, pauseXlet, startXlet.
 void initMobilet(MobiletContext ctx);  // maybe make this an abstract class instead ?

 void setForeground(boolean fg);  // should be (a) lazy (b) throw exceptions ?
boolean getForeground();
String          getName();
void          setName(String name);
}
```

```
import java.awt.*;
import java.applet.*;
public class MobiletApplet extends Frame {
MobiletManager  fMobiletManager;
 Dimension      fDimension;
MobiletApplet() {
setSize(400,400);
setVisible(true);
 init( );
start( );
 }
public void init( ) {
}
public void start( ) {
fDimension = getSize( );
fMobiletManager = new AMobiletManager( fDimension, new Dimension(350, 350));
MobiletRegistry registry = fMobiletManager.getMobiletRegistry( );
Panel   fPanel = fMobiletManager.getMobiletManagerPanel();
add(fPanel);
fPanel.setVisible(true);
 fPanel.validate( );
 repaint( );

Mobilet    mobilet = new AOLInstantMessengerServer( );

MobiletID    id = registry.addMobilet(mobilet);
 fMobiletManager.initMobilet(id);
 fMobiletManager.startMobilet(id);
}
public static void main(String[] args) {
MobiletApplet ma = new MobiletApplet( );


}
}
import java.net.URL;
public abstract interface MobiletBrowser extends Mobilet {
void initMobiletBrowser(URL url);
```

```
}
import java.awt.Dimension;
public abstract interface MobiletContext extends XletContext {
// inherits destroyed, getXletPropert(java.lang.String key), paused, resumeRequest
Dimension getDimension( ); // gets the dimension of the mobilet's drawing surface.
 MobiletPanel getMobiletPanel( );
MobiletRegistry  getMobiletRegistry( );
Mobilet  getMobilet( );
void   setInited(boolean inited);
void setForeground(boolean fg);
}
mport java.net.URL;
public abstract interface MobiletLoader {
Mobilet  load(URL url);
 void unload(Mobilet mobilet);
}
// MobiletManager should be extended to make sure that we can make a "VISIBLE" Mobilet
// manager
import java.awt.Dimension;
import java.awt.Panel;
public abstract interface MobiletManager {
Dimension  getDisplayDimension( );
Dimension  getMaxMobiletDimension( );
Panel getMobiletManagerPanel( );
MobiletRegistry getMobiletRegistry( );
void  setForeground(Mobilet mobilet);
void  setForeground(MobiletContext ctx, boolean fg);
 Mobilet  getCurrentForeground( );
// boolean  getForeground(Mobilet mobilet);
void   destroyed(MobiletContext ctx);
void   paused(MobiletContext ctx);
void  initMobilet(MobiletID id);
void startMobilet(MobiletID id);
 // iterator for getting all mobilets.
}
import java.awt.Panel;
public class MobiletPanel extends Panel {
```

```
MobiletID fMobiletID;
MobiletPanel(MobiletID mobiletID) {
fMobiletID = mobiletID;
}
}
public abstract interface MobiletRegistry {
MobiletID addMobilet(Mobilet mobilet);
// should name be an URL instead ?
MobiletID  getMobiletID(String name);
MobiletID  getMobiletID(Mobilet mobilet);  // a factory model ?
Mobilet   getMobilet(MobiletID id);
boolean    destroyMobiletID(Mobilet mobilet, MobiletID id);
// an iterator or something that returns a collection
}
public interface Xlet {
/**
    * Signals the Xlet to initialize itself and enter the
    * <i>Paused</i> state.
    * The Xlet shall initialize itself in preparation for providing service.
    * It should not hold shared resources but should be prepared to provide
    * service in a reasonable amount of time. <p>
    * An <code>XletContext</code> is used by the Xlet to access
    * properties associated with it's runtime environment.
    * After this method returns successfully, the Xlet
    * is in the <i>Paused</i> state and should be quiescent. <p>
   * <b>Note:</b> This method shall only be called once.<p>
    *
    * @parameter ctx XletContext This Xlet's XletContext
    * @exception com.sun.javax.tv.XletStateChangeException
    * @see com.sun.javax.tv.xlet.XletContext
    */
public void initXlet(XletContext ctx) throws XletStateChangeException;
/**
    * Signals the Xlet to start providing service and
    * enter the <i>Active</i> state.
    * In the <i>Active</I> state the Xlet may hold shared resources.
   * The method will only be called when
```

```
     * the Xlet is in the <i>paused</i> state.
     * <p>
     * Two kinds of failures can prevent the service from starting,
     * transient and non-transient.  For transient failures the
     * <code>XletStateChangeException</code> exception should be thrown.
     * For non-transient failures the <code>XletContext.done</code>
     * method should be called with an error indication (TBD).
     *
     *
     * @exception XletStateChangeException  is thrown if the Xlet
     *                cannot start providing service.
     */

    public void startXlet() throws XletStateChangeException;
   /**
    *
     * Signals the Xlet to stop providing service and
     * enter the <i>Paused</i> state.
     * In the <i>Paused</i> state the Xlet must stop providing
     * service, and might release all shared resources
     * and become quiescent. This method will only be called
     * called when the Xlet is in the <i>Active</i> state. <p>
     *
    */
    public void pauseXlet( );
  /**
     * Signals the Xlet to terminate and enter the <i>Destroyed</i> state.
     * In the destroyed state the Xlet must release
     * all resources and save any persistent state. This method may
     * be called from the <i>Loaded</i>, <i>Paused</i> or
     * <i>Active</i> states. <p>
     * Xlets should
     * perform any operations required before being terminated, such as
     * releasing resources or saving preferences or
     * state. <p>
     *
     * <b>NOTE:</b> The Xlet can request that it not enter the <i>Destroyed</i>
```

```
     * state by throwing an <code>XletStateChangeException</code>. This
     * is only a valid response if the <code>unconditional</code>
    * flag is set to <code>false</code>. If it is <code>true</code>
     * the Xlet is assumed to be in the <i>Destroyed</i> state
     * regardless of how this method terminates. If it is not an
     * unconditional request, the Xlet can signify that it wishes
     * to stay in its current state by throwing the Exception.
     * This request may be honored and the <code>destroy()</code>
     * method called again at a later time.
     *
     *
     * @param boolean unconditional If <code>done</code> is true when this
     * method is called, requests by the Xlet to not enter the
     * destroyed state will be ignored.
     *
     * @exception XletStateChangeException is thrown if the Xlet
     *                wishes to continue to execute (Not enter the <i>Destroyed</i>
     *         state).
     *         This exception is ignored if <code>unconditional</code>
    *         is equal to <code>true</code>.
     *
     *
     */
    public void destroyXlet(boolean unconditional)
                throws XletStateChangeException;
}
public interface XletContext {

    /**
     *
     * Signals that the Xlet has entered itself into the
     * <i>Destroyed</i> state. The application manager should update
     * the state to <i>Destroyed</i> without calling the Xlet's
     * <code>destroy</code> method. The Xlet must perform the same operations
    * (clean up, releasing of resources etc.) it would have if the
     * <code>destroy( )</code> was called.
     *
```

```
    */
public void destroyed( );
/**
    *Signals that the Xlet does not want to be active and  has
    * entered the <i>Paused</i> state.  This method can  only be
    * invoked when the Xlet is in the <i>Active</i> state. <p>
    *
    * If an Xlet calls <code>paused</code>, in the
    * future it may be asked to enter <i>Destroyed</i> state or
    * the <i>Active</i> state again.
    *
    */

   public void paused( );
/**
    * Provides an Xlet with a mechanism to retrieve named
    * properties from the XletContext.
    *
    * @parameter key The name of the property
    * @return A reference to an object representing the property.
    *              <code>null</code> is returned if no value is available for key.
    */
public Object getXletProperty(String key);
/**
    * Provides the Xlet with a mechanism to indicate that it is
    * interested in entering the <i>Active</i> state. Calls to
    * this method can be used by an application manager to determine which
    * Xlets to move to <i>Active</i> state.
    */

  public void resumeRequest( );
}
public interface XletLifeCycle {

   /**
    * Initialize the Xlet. This method is a signal to the Xlet that
    * initialize itself such that it prepared to provide it's
```

```
   * <i>Service</i> in a reasonable amount of time. An
    * <code>XletContext</code> object is passed in with this
   * method. This object can be used by the Xlet to access
   * properties associated with it's environment as well as having
    * a way to signal back to the <i>Application Manager</i> that
    * it is changing state.
    *
    * @parameter ctx XletContext This Xlet's Xlet Context
    */
   public void init(XletContext ctx);

   /**
    * The Xlet is moved to the <i>In Service</i> state when this
    * method completes. The Xlet is now expected to be providing
    * service.
    */
public void enterService( );
/**
    * The <code>leaveService</code> callback signals the Xlet to
    * to stop providing service. Then when the callback returns the
    * Xlet is in the <i>Out of Service</i> state.
    */
   public void leaveService( );

   /**
   * This method is a signal to the Xlet that it's no longer needed
    * and that it will shortly be purged from the system. Xlets should
   * perform any operations required before being terminated such
    * as releasing resources or saving preferences or state.
   */
   public void destroy( );
}
public class XletStateChangeException extends Exception {
/**
    * Constructs an exception with no specified detail message.
   */
public XletStateChangeException( ){ }
```

```
  /**
   * Constructs an exception with the specified detail message.
   *
   * @param s the detail message
   */
public XletStateChangeException(String s){}

}
import java.awt.Dimension;
import java.awt.Panel;
import java.awt.Color;
import java.awt.FlowLayout;
public class AMobiletManager implements MobiletManager {
Dimension      fDisplayDimension;
Dimension      fMaxMobiletDimension;
Mobilet        fCurrentForegroundMobilet;
MobiletRegistry fMobiletRegistry;
Panel    fPanel;
  AMobiletManager(Dimension displayDimension, Dimension maxMobiletDimension) {

   fDisplayDimension    = displayDimension;
   fMaxMobiletDimension = maxMobiletDimension;
// Create a registry;
  fMobiletRegistry = new AMobiletRegistry(this);
// Create a display;
fPanel = new Panel( );
fPanel.setLayout(new FlowLayout( ));
fPanel.setSize(fDisplayDimension);
// Make it visible (optional .. may make it an interface feature
// in order to make it work better with other native apps.
fPanel.setVisible(false);

// This canvas functionality will be highly dependent on target
// should use properties here.
// one might conceive of a single canvas with different
```

```
// clipping areas in order to manage different mobilets, guide tools etc.
// in order to save "bitmap" space.  AWT implementation itself may choose
// to do so.  For now, lets keep it simple.
}
public Dimension  getDisplayDimension( ) {
return fDisplayDimension;
 }
public Dimension getMaxMobiletDimension() {
return fMaxMobiletDimension;
}
 public Panel getMobiletManagerPanel( ) {
  return fPanel;
 }
 public Mobilet getCurrentForeground( ) {

  return fCurrentForegroundMobilet;
 }

 public void setForeground(Mobilet mobilet) {  // define exceptions ...

  //  first check if mobilet registered
if (fMobiletRegistry.getMobiletID(mobilet) == null) {
fMobiletRegistry.addMobilet(mobilet);
}
  if (fCurrentForegroundMobilet == mobilet) {
return;
}
// need resource mgt:  need to put previous mobilet into background.
fCurrentForegroundMobilet = mobilet;
}
public void setForeground(MobiletContext ctx, boolean fg){
MobiletPanel mp = ctx.getMobiletPanel( );
// mp.setVisible(fg);
// mp.validate( );
// System.out.println("mobilet panel = " + mp);
// mp.setBackground(Color.red);
if (fg) {
```

```
    fPanel.add(mp);
mp.setLocation(60,60);
  }
  else
 fPanel.remove(mp);
fPanel.setVisible(true);
fPanel.repaint( );
System.out.println(fPanel);
 System.out.println("mobilet panel again = " + mp);
fPanel.list(System.out);


  }
public MobiletRegistry getMobiletRegistry( ) {
return fMobiletRegistry;
 }
public void  initMobilet(MobiletID id) {

Mobilet mobilet = id.getMobilet( );
MobiletContext ctx = new AMobiletContext(this, mobilet);
id.setContext(ctx);
mobilet.initMobilet(ctx);
ctx.setInited(true);
}

public void startMobilet(MobiletID id) {
Mobilet mobilet = id.getMobilet( );
MobiletContext  ctx = id.getContext( );
  try {
    mobilet.startXlet( );
  } catch (XletStateChangeException e) {
     e.printStackTrace( );
  }
  ctx.setInited(true);
}
public void pauseMobilet(MobiletID id) {
  Mobilet mobilet = id.getMobilet( );
```

```
  mobilet.pauseXlet( );
}
public void destroyed(MobiletContext ctx) {
}
public void paused(MobiletContext ctx){

 }
}
import java.awt.*;
import java.util.Properties;
public class AMobiletContext implements MobiletContext {
boolean   fDestroyed = false;
boolean   fPaused   = false;
boolean   fActive   = false;

boolean   fInited   = false;
boolean   fStarted  = false;
Properties fProperty;
Mobilet   fMobilet;
AMobiletManager fMgr;
MobiletPanel   fPanel;
Dimension     fDimension;
AMobiletContext(AMobiletManager mgr, Mobilet mobilet) {
 fMgr = mgr;
fMobilet = mobilet;
fProperty = new Properties( );
fPanel = new MobiletPanel(mgr.getMobiletRegistry( ).getMobiletID(mobilet));
fDimension = mgr.getMaxMobiletDimension( );
fPanel.setSize(fDimension);
fPanel.setLayout(new BorderLayout ( ) );
  }
public Dimension getDimension( ) {
  return fDimension;
  }
public MobiletPanel getMobiletPanel( ) {
   return fPanel;
 }
```

```
 public MobiletRegistry getMobiletRegistry( ) {
return fMgr.getMobiletRegistry( );
 }
public void destroyed( ) {
if (!fDestroyed && (fPaused  || fActive)) {
fMgr.destroyed(this);
fPaused = false;
fActive = false;
fDestroyed = true;
}
}
public void paused() {
 if (!fDestroyed && !fPaused && fActive) {
 fMgr.paused(this);
fPaused = true;
fActive = false;
}
}
 public Object getXletProperty(String key){
  return fProperty.getProperty(key);
}
public void resumeRequest() {
  if (!fDestroyed && !fActive && fPaused) {
fMgr.setForeground(this, true);
fActive = true;
fPaused = false;
}
}
public void setInited(boolean inited) {
fInited = inited;
}
 public void setStarted(boolean started) {
fStarted = started;
}
public void setForeground(boolean fg) {
System.out.println("AMobiletContext.setForeground " + fg);
  fMgr.setForeground(this, fg);
```

```
 }
 public Mobilet  getMobilet( ){
return fMobilet;
 }
}
public class  AMobiletID implements MobiletID{
String   fMobiletName;

 Mobilet   fMobilet;
MobiletContext  fCtx;
AMobiletID(Mobilet mobilet) {
fMobilet = mobilet;
fMobiletName = fMobilet.getName();
}
public String getName( ) {
return fMobiletName;
}
public boolean isActive( ) {
return true;
}
public boolean isEqual(MobiletID other) {
return this.fMobiletName.equals(other.getName( ));
}
public Mobilet getMobilet( ) {
return fMobilet;
 }
public void setContext(MobiletContext ctx) {
fCtx = ctx;
}
public MobiletContext getContext( ) {
 return fCtx;
}
}
import java.util.Hashtable;
import java.util.Enumeration;
public class AMobiletRegistry implements MobiletRegistry{
Hashtable  fMobiletHash;
```

```
        MobiletManager fMgr;
         AMobiletRegistry(MobiletManager mgr) {
        fMgr = mgr;
        fMobiletHash = new Hashtable( );

         }
        public MobiletID addMobilet(Mobilet mobilet) {
        AMobiletID  mobiletID = new AMobiletID(mobilet);
        fMobiletHash.put(mobilet.getName( ), mobiletID);
        return mobiletID;
         }
        public MobiletID  getMobiletID(String name) {
        return (MobiletID) fMobiletHash.get(name);
         }
        public MobiletID  getMobiletID(Mobilet mobilet){
        Enumeration keys = fMobiletHash.keys( );
        AMobiletID  id;
        String key;
         while (keys.hasMoreElements( )) {
          key = (String) keys.nextElement( );
        id = (AMobiletID) fMobiletHash.get(key);
        if (id.getMobilet( ) == mobilet)
        return id;
         }
        return null;
         }
        public Mobilet    getMobilet(MobiletID id) {
        return id.getMobilet( );
         }
        public boolean   destroyMobiletID(Mobilet mobilet, MobiletID id){
          fMobiletHash.remove(id.getName( ));
        return true;
         }
        // an iterator or something that returns a collection
}
        import java.awt.*;
        import java.awt.event.*;
```

```
import java.net.*;
import java.io.*;
public class AOLInstantMessengerServer extends
AOLInstantMessenger implements Mobilet, ActionListener{
int  fServerPortNumber = kPort;
MobiletContext  fCtx;
Button          fButton;
TextArea        fTx;
MobiletPanel    fPanel;
static final int timeout = 1000;  // 1000 milliseconds = 1 second
boolean         fKeepRunning;
String          fMessages;
 AOLInstantMessengerServer(int port) {
super(port);
try {
   fSock.setSoTimeout(timeout);
 }
 catch (Exception e){
e.printStackTrace( );
 }
 }
 AOLInstantMessengerServer( ) {
  super(kPort);
try {
 fSock.setSoTimeout(timeout);
  }
 catch (Exception e) {
 e.printStackTrace( );
   }
 }
public void initXlet(XletContext ctx) {
  // add a button to the panel
  // make this an observer
 fButton = new Button("OK");
 fButton.addActionListener(this);

  fCtx = (MobiletContext) ctx;
```

```
  fButton.setSize(30,30);
  fButton.setVisible(true);
  fCtx.getMobiletPanel( ).setLayout(new BorderLayout());
 fCtx.getMobiletPanel( ).add(fButton, BorderLayout.SOUTH);
  fCtx.getMobiletPanel( ).setVisible(true);
 }
 public void initMobilet(MobiletContext ctx) {
 initXlet(ctx);  // should actually check if inited already.
 fCtx = ctx;
 }
public void setForeground(boolean fg) {
fCtx.setForeground(fg);
}
 synchronized public void startXlet ( ) {
fKeepRunning = true;
  while (fKeepRunning) {
try {
DatagramPacket  message = new DatagramPacket(new byte[1024], 1024);
fSock.receive(message);
System.out.println("message rcvd:" + new String(message.getData()));
System.out.println(fMessages);
if (fTx == null)
 fTx = new TextArea(new String(message.getData()));
else
 // TextArea tx = new TextArea(new String(message.getData()));
 fTx.append("\n" + new String(message.getData()) );
fTx.setSize(100,100);
fTx.setVisible(true);
fCtx.getMobiletPanel().add(fTx, BorderLayout.NORTH);
fTx.setLocation(40,40);
//fTx = tx;
setForeground(true);

try {
 System.out.println("About to wait");
 wait( );
}
```

```
        catch (InterruptedException e) {

         // ignore
        }
        // fCtx.setForeground();
          }
    catch (java.io.InterruptedIOException e) {
        // do nothing since it is probably a datagram rcv timeout
          }
    catch (Exception e) {
        e.printStackTrace();
          }
}
        }
        synchronized public void actionPerformed(ActionEvent evt) {
         fTx.setVisible(false);
        fCtx.getMobiletPanel().remove(fTx);
        setForeground(false);
        notify( );
        // fTx = null;
        }
        public void pauseXlet( ){
         fKeepRunning = false;
          }
        public String getName( ) {
           return "AOLInstantMessenger";
          }
        public void setName(String name) {
         }
         public void destroyXlet(boolean doit) {

        }
        public boolean getForeground( ) {
          return true; // hack;
        }
        }
```

Thus, an application framework for mobile devices have been described in conjunction with one or more specific embodiments. The invention is defined by the claims and their full scope of equivalents.

Claims

1. An application framework for mobile devices comprising:

a multi-tier architecture comprising a first tier capable of processing device-independent applications, a third tier providing a plurality of services to said first tier, a second tier for preprocessing communications between said first tier and said third tier thereby reducing processing requirements on said first tier;

a plurality of peer-to-peer communication layers between said third tier and said first tier through said second tier, said second tier providing protocol translation between said third tier and said first tier.

2. The application framework of claim 1, wherein said plurality of peer-to-peer layers comprises:

at least one physical data link layer
a network layer;
a transport layer;
a session layer;
a presentation layer; and
an applications layer.

3. The application framework of claim 2, wherein said at least one physical data link layer comprises landline communication between said third tier and said second tier, and wireless communication between said second tier and said first tier.

4. The application framework of claim 2, wherein said network layer uses Internet Protocol communication between said third tier and said second tier, and wireless applications protocol between said second tier and said first tier.

5. The application framework of claim 2, wherein said transport layer uses transport control protocol between said third tier and said second tier, and wireless applications protocol between said second tier and said first tier.

6. The application framework of claim 2, wherein said session layer uses hypertext transport protocol between said third tier and said second tier and amongst services in said third tier, and wireless applications protocol between said second tier and said first tier.

7. The application framework of claim 2, wherein said presentation layer uses a markup language between said third tier and said second tier, and a wireless markup language between said second tier and said first tier.

8. The application framework of claim 2, wherein said application layer prepares graphical data for presentation, said graphical data being available in any suitable graphical format and communicated from said third tier to said second tier, said second tier converting said graphical data to a wireless graphics format for transmission to said first tier.

9. The application framework of claim 1, wherein said first tier is a wireless device.

10. The application framework of claim 9, wherein said wireless device is a cellular phone.

11. The application framework of claim 9, wherein said wireless device is a palm device.

12. The application framework of claim 9, wherein said wireless device includes a software architecture comprising:

a real-time operating system layer;

a virtual machine layer having at least one system class; and

an application layer.

13. The application framework of claim 12, wherein said real-time operating system layer comprises: a wireless small device operating system; a plurality of linking and networking application programming interfaces; and an object for updating and installing software in said wireless device.

14. The application framework of claim 12, wherein said application layer comprises:
a platform specific framework object class;
a platform independent framework object class; and
at least one application object class.

15. The application framework of claim 14, wherein said at least one application object class may operate in any of a plurality of states, wherein said plurality of states comprises an initialization state, a background state, a foreground state, a destroy state, and a paused state.

16. The application framework of claim 15, further comprising a manager object for managing each of said at least one application object class in said plurality of states.

17. An application framework for mobile devices comprising:

a multi-tier architecture comprising a client tier having a virtual machine capable of processing device-independent applications, a server tier providing a plurality of services to said client tier in the form of said device-independent applications, a gateway tier for preprocessing communications between said client tier and said server tier thereby reducing processing requirements on said client tier;

a plurality of peer-to-peer communication layers between said server tier and said client tier through said gateway tier, said gateway tier providing protocol translation between said server tier and said client tier;

a manager object in said client tier for managing said device-independent applications, each of said device-independent applications having a plurality of states, wherein said

plurality of states comprises an initialization state, a background state, a foreground state, a destroy state, and a paused state.

18. A multi-tier system for providing vendor-neutral communication to mobile devices comprising:

a client device having a virtual machine capable of processing device-independent applications, a plurality of servers providing a plurality of services to said client device in the form of said device-independent applications, a gateway for preprocessing communications between said client device and said plurality of servers thereby reducing processing requirements on said client device;

a plurality of peer-to-peer communication layers between said plurality of servers and said client device through said gateway, said gateway providing protocol translation between said plurality of servers and said client device;

a manager object in said client device for managing said device-independent applications, each of said device-independent applications having a plurality of states, wherein said plurality of states comprises an initialization state, a background state, a foreground state, a destroy state, and a paused state.

Figure 1

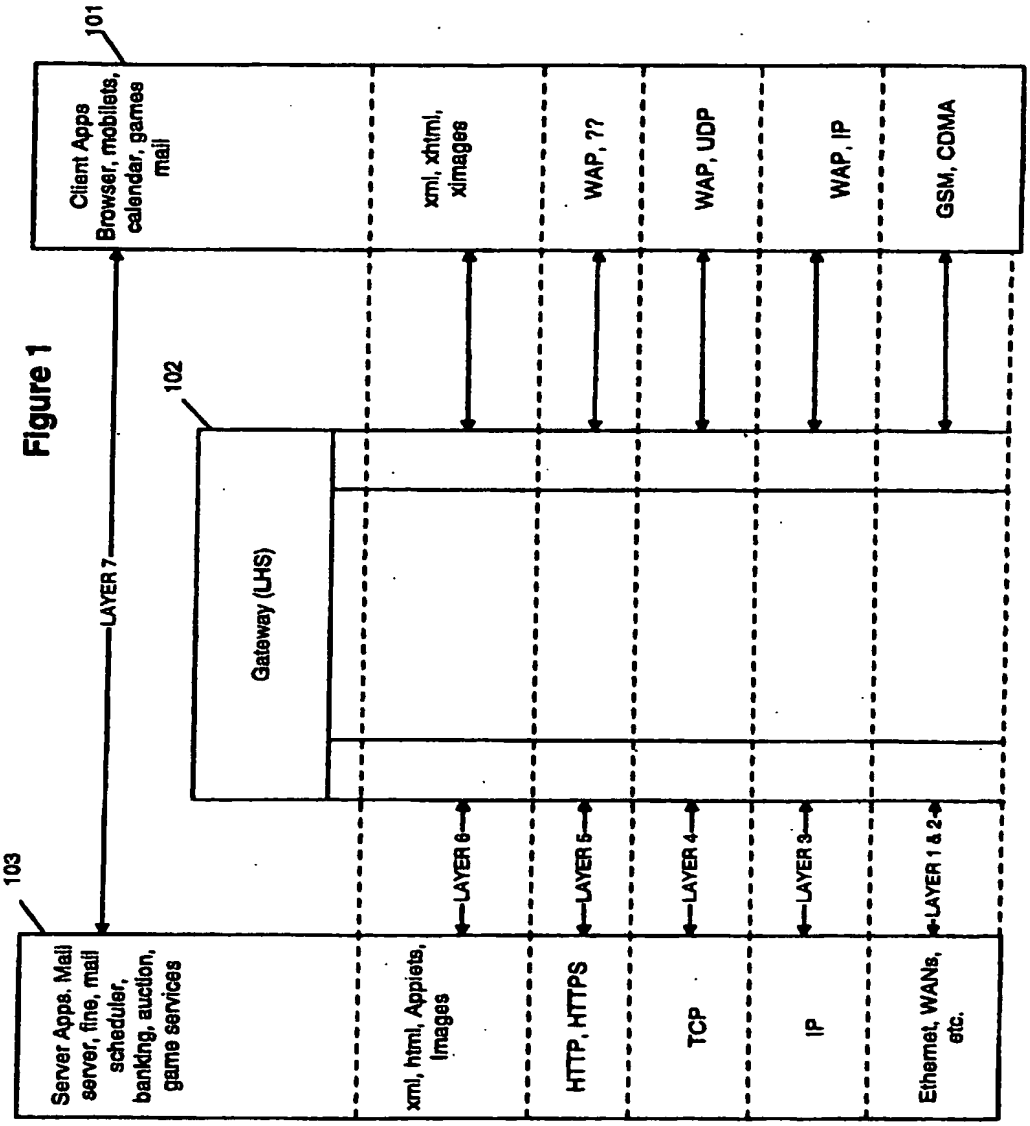
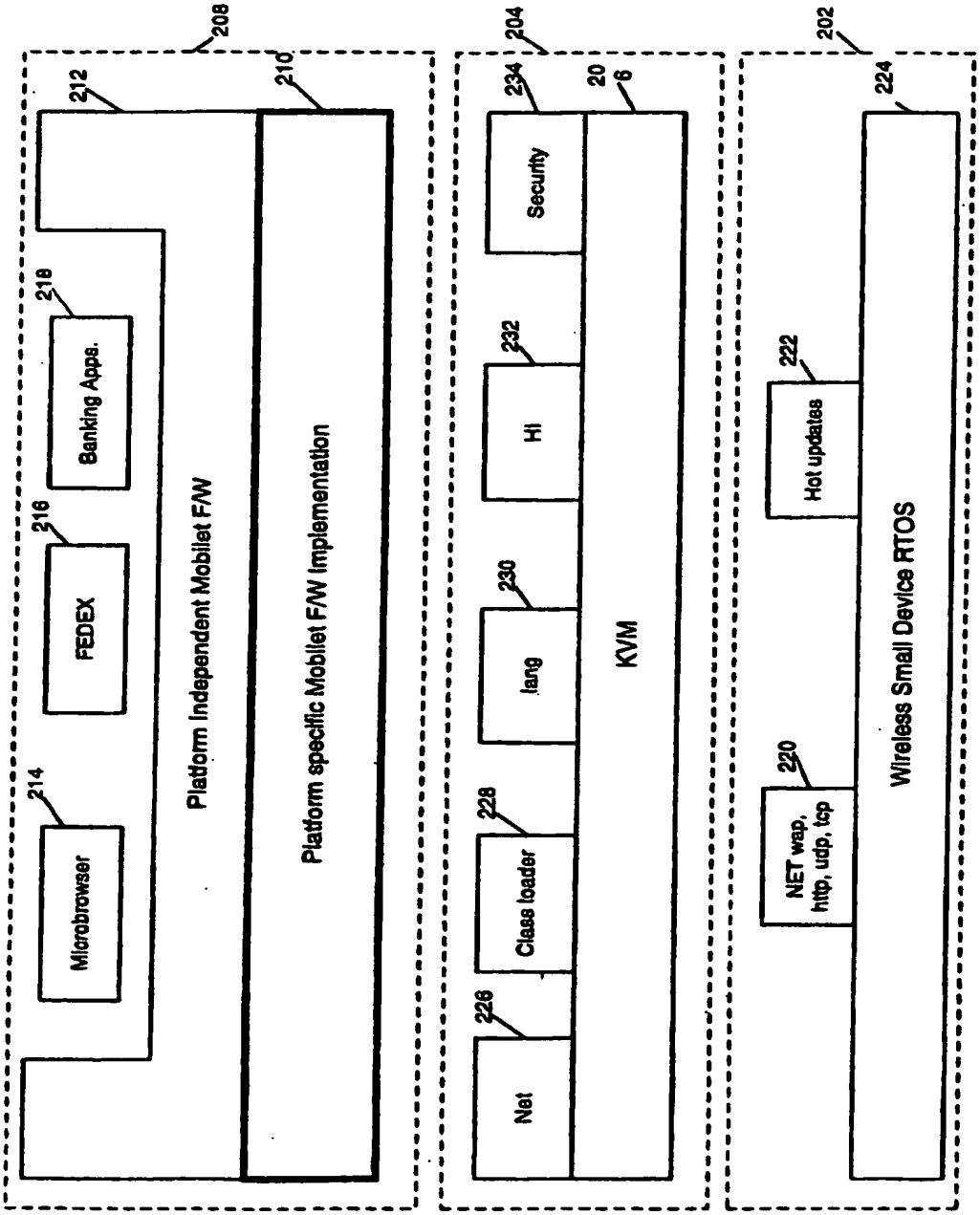

101
Client Apps
Browser, mobilets, calendar, games mail
xml, xhtml, ximages
WAP, ??
WAP, UDP
WAP, IP
GSM, CDMA
102
Gateway (LHS)
LAYER 7
LAYER 6
LAYER 5
LAYER 4
LAYER 3
LAYER 1 & 2
103
Server Apps. Mail server, fine, mail scheduler, banking, auction, game services
xml, html, Applets, Images
HTTP, HTTPS
TCP
IP
Ethernet, WANs, etc.

Figure 2


208
212
210
214
Microbrowser
216
FEDEX
218
Banking Appa.
Platform Independent Mobilet F/W
Platform specific Mobilet F/W Implementation
204
234
Security
232
Hi
230
lang
228
Class loader
226
Net
206
KVM
202
224
Wireless Small Device RTOS
222
Hot updates
220
NET wap, http, udp, tcp

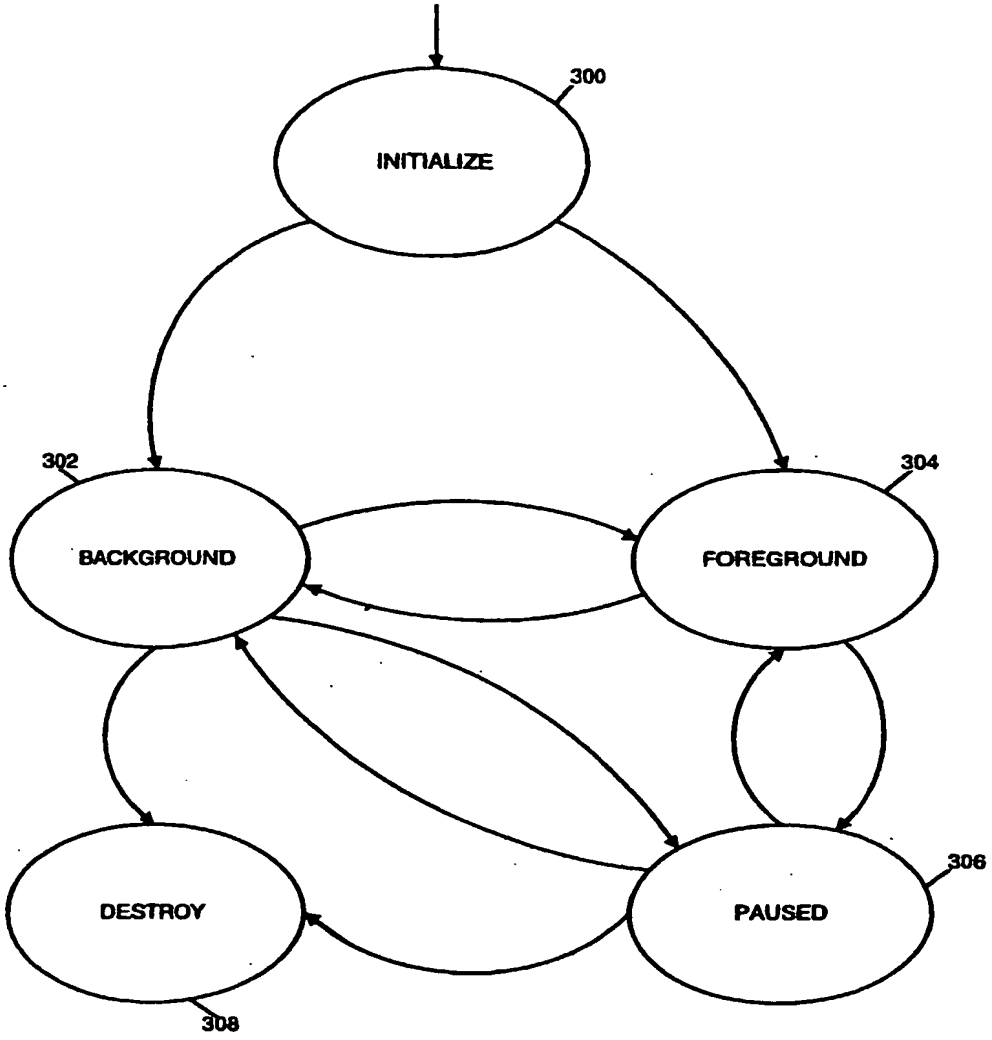

300
INITIALIZE
302
BACKGROUND
304
FOREGROUND
306
PAUSED
DESTROY
308

Figure 3

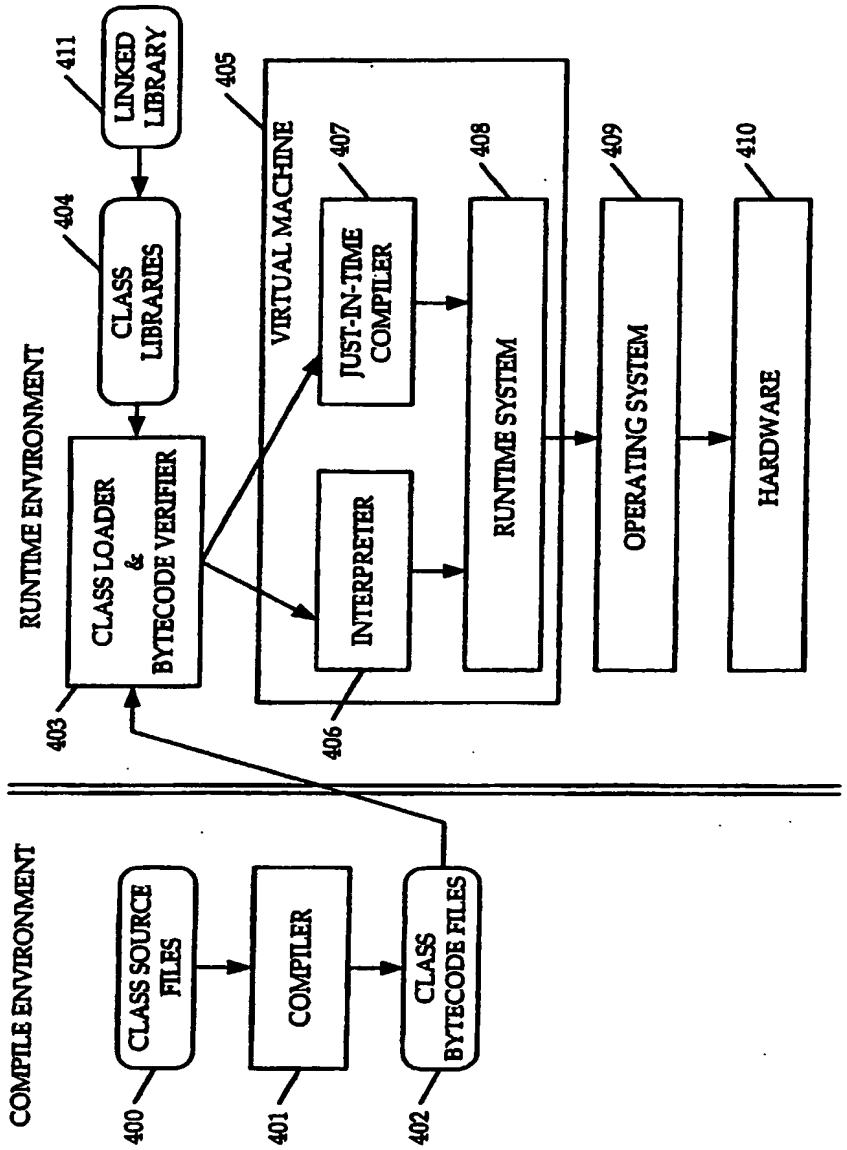
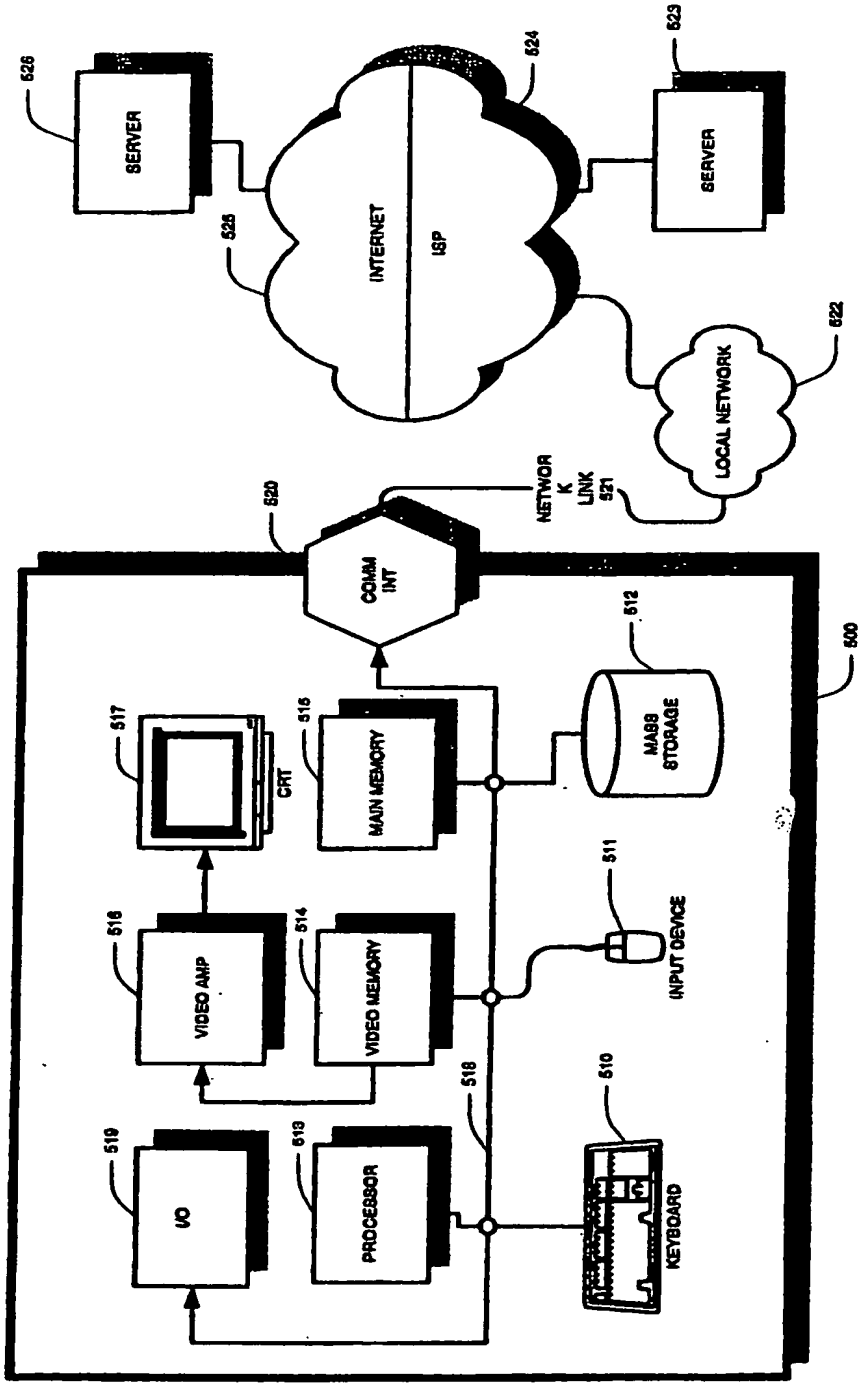

COMPILE ENVIRONMENT
CLASS SOURCE FILES
400
COMPILER
401
CLASS BYTECODE FILES
402
RUNTIME ENVIRONMENT
CLASS LOADER & BYTECODE VERIFIER
403
CLASS LIBRARIES
404
LINKED LIBRARY
411
VIRTUAL MACHINE
405
INTERPRETER
406
JUST-IN-TIME COMPILER
407
RUNTIME SYSTEM
408
OPERATING SYSTEM
409
HARDWARE
410

FIGURE 4

Figure 5


526
SERVER
524
INTERNET
ISP
523
SERVER
525
522
LOCAL NETWORK
NETWOR
K
LINK
521
520
COMM
INT
500
512
MASS
STORAGE
517
CRT
515
MAIN MEMORY
511
INPUT DEVICE
516
VIDEO AMP
514
VIDEO MEMORY
518
510
KEYBOARD
519
I/O
513
PROCESSOR

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☒ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.